Input/Output

There are not many new features in Fortran for I/O, so this provides an opportunity to review some of the basic concepts.

The input/output statements are

- read
- print
- write
- open
- close
- inquire
- backspace
- endfile
- rewind

Next slide

The `open` and `close` statements deal with the connection between an input/output unit and a file;

The `inquire` statement provides the means to find out things about a file or unit.

The `read`, `write`, and `print` statements are the ones that do the actual data transfer.

The `backspace`, `endfile`, and `rewind` statements affect the position of the file.

Learn more about the `open` statement.
Learn more about the `close` statement.
Learn more about the `inquire` statement.
Learn more about the data transfer statements.
Learn more about the file positioning statements.

Previous slide Next slide

Records

There are two kinds of records, *data records* and *end-of-file records*. A data record is a sequence of values.

```
 ___    ___    ___           ___
|   |  |   |  |   |         |   |
|   |  |   |  |   | . . .   |   |
|___|  |___|  |___|         |___|
```

Previous slide Next slide

The values in a data record may be represented in one of two ways: formatted or unformatted. If the values are characters readable by a person, each character is one value and the data is *formatted*.

```
    ____     ____     ____
   |    |   |    |   |    |
   | 6  |   | 1  |   | 1  |
   |____|   |____|   |____|
```

Previous slide Next slide

Unformatted data consists of values usually represented just as they are stored in computer memory.

```
  ____________    ____________
 |            |  |            |
 |  00000110  |  |  00001011  |
 |____________|  |____________|
```

Previous slide Next slide

The other kind of record is the *end-of-file record*, which, at least conceptually, contains no values and has no length. There can be at most one end-of-file record in a file and it must be the last record of a file. It is used to mark the end of a file.

Previous slide Next slide

Files

A *file* is a collection of records.

```
  ____________
 |            |
 |____________|
  ____________
 |            |
 |____________|
     .  .  .
  ____________
 |            |
 |____________|
```

The records of a file must be either all formatted or all unformatted, except that the file may contain an endfile record as the last record.

Previous slide Next slide

Two kinds of files are

- External files
- Internal files

An external file usually is stored on a peripheral device, such as a tape, a disk, or a computer terminal.

Internal files are stored in memory as values of character variables.

Previous slide Next slide

File Access Methods

There are two access methods for external files:

- Sequential access
- Direct access

Previous slide Next slide

Sequential access to the records in the file begins with the first record of the file and proceeds sequentially to the second record, and then to the next record, record by record.

```
  __________
 |          |
 | record 1 |
 |__________|
  __________
 |          |
 | record 2 |
 |__________|
    . . .
  __________
 |          |
 | record n |
 |__________|
```

Previous slide Next slide

When a file is accessed directly, the records are selected by record number.

```
 _____________
|             |
| record 47   |
|_____________|
 _____________
|             |
| record 13   |
|_____________|
```

The following rules apply when accessing a file directly:

1. If a file is to be accessed directly, all of the records must be the same length.
2. It is not possible to delete a record using direct access.
3. List-directed input/output and nonadvancing input/output are prohibited.
4. An internal file must be accessed sequentially.

Learn more about records and files.

Previous slide Next slide

File Position

Each file being processed by a program has a *position*. During the course of program execution, records are read or written, causing the position of the file to change. Also, there are Fortran statements that cause the position of a file to change; an example is the `backspace` statement.

Previous slide Next slide

The *initial point* is the point just before the first record. The *terminal point* is the point just after the last record. If the file is empty, the initial point and the terminal point are the same.

```
initial point --->   __________
                    |          |
                    |__________|
                     __________
                    |          |
                    |__________|
                       . . .
                     __________
                    |          |
                    |__________|
terminal point --->
```

Previous slide Next slide

A file may be positioned between records.

```
              _____________
             |             |
             |_____________|
      --->    _____________
             |             |
             |_____________|
                 .  .  .
              _____________
             |             |
             |_____________|
```

Previous slide Next slide

There may be a current record during execution of an input/output statement or after completion of a nonadvancing input/output statement.

```
                     __________
                    |          |
                    |__________|
                     __________
current record ---> |          |
                    |__________|
                       . . .
                     __________
                    |          |
                    |__________|
```

Learn more about file position.

Previous slide Next slide

When there is a current record, the file is positioned at the initial point of the record, the terminal point of the record, or between values in a record.

```
     ___    ___    ___         ___    ___
    |   |  |   |  |   |       |   |  |   |
    |   |  |   |  |   | . . . |   |  |   |
    |___|  |___|  |___|       |___|  |___|
   ^             ^                        ^
   |             |                        |
   |             |                        |
initial point  between values  terminal point
```

Previous slide Next slide

Advancing and Nonadvancing I/O

Advancing input/output is record oriented. Completion of an input/output operation always positions the file at the end of a record. Nonadvancing input/output is character oriented. After reading and writing, it is possible that the file position is between characters within a record.

Nonadvancing input/output is indicated by `advance= "no"` in the I/O control list. It is restricted to use with external sequential formatted files.

Learn more about nonadvancing input/output.

Previous slide Next slide

```
program char_count
   implicit none
   integer, parameter :: end_of_record = -2
   integer, parameter :: end_of_file = -1
   character (len = 1) :: c
   integer :: count, ios
```

Previous slide Next slide

```
   count = 0
   do
      read (*, "(a)", advance = "no",  &
            iostat = ios) c
      if (ios == end_of_record) then
         cycle
      else if (ios == end_of_file) then
         exit
      else
         count = count + 1
      end if
   end do

   print *, "The number of characters " //  &
         "in the file is ", count
end program char_count
```

Previous slide Next slide

Exercise

1. Determine the ratio of vowels to consonants in an arbitrary file of text.

Learn more about the character data type.

Previous slide Next slide

The en Edit Descriptor

`en` is the engineering edit descriptor. On input, the form is the same as for `f` editing. On output, the number is in the form of engineering notation, where the exponent is divisible by three and the absolute value of the mantissa is greater than or equal to 1 and less than 1000, except when the output value is 0.

Learn more about data edit descriptors.
Learn more about control edit descriptors.

Previous slide Next slide

```
Internal value     Output field using ss, en12.3
________________________________________________
     6.421                6.421E+00
     -.5               -500.000E-03
      .0217              21.700E-03
  4721.3                  4.721E+03
```

Previous slide Next slide

The es Edit Descriptor

`es` is the scientific edit descriptor. On input, the form is the same as for `f` editing. On output, the number is in the form of scientific notation; the absolute value of the mantissa is greater than or equal to 1 and less than 10, except when the output value is 0.

Previous slide Next slide

```
Internal value     Output field using ss, es12.3
________________________________________________
     6.421                  6.421E+00
     -.5                   -5.000E-01
      .0217                 2.170E-02
  4721.3                    4.721E+03
```

Previous slide Next slide

Zero Field Width (F95)

An edit descriptor with zero field width indicates that as few columns as possible should be used to produce the result.

```
price = 44.23
print "(a, f0.2, a)", "The price is $", price, "."
```

produces the output:

```
 The price is $44.23.
```

Previous slide Next slide

Namelist Input/Output

Namelist input/output accomplishes two different things. It provides a mechanism for naming the objects that may occur in an input/output list (as you might expect), but only in namelist input/output statements. It also provides a mechanism whereby the input data may contain values for only a portion of the variables in the input list. Here are two very simple examples.

Previous slide Next slide

```
real a (3)
character (len = 3) char
complex x
logical ll
namelist / token / i, a, char, x, ll
read (*, nml = token)
```

If the input record is

```
&token a(1:2)=2*1.0 ll=t char="nop" x=(2.4,0.0)/
```

results of the `read` are

```
  I       unchanged
a (1)       1.0
a (2)       1.0
a (3)     unchanged
char        nop
X         (2.4, 0.0)
ll           t
```

Previous slide Next slide

To illustrate namelist output:

```
namelist / turn / i, x
write (*, nml = turn)
```

produces the output record:

```
&turn  i = 20  x = 2.468 /
```

Learn more about namelist editing.

Previous slide Next slide

This could be done in Fortran 77, but a frequently asked question is: How can I convert a character string into a number? We have done this with a function, but the simple way is to use an internal file.

The following example opens twenty files named `file091.txt`, `file092.txt`, ... `file110.txt`, writes an integer into each, and closes them with status="keep".

Previous slide Next slide

```
program open_files

   implicit none
   integer :: n, ios
   character (len=11) :: &
      file_name = "file???.txt"

   do n = 91, 110
      write (file_name(5:7), "(i3.3)") n
      open (file=file_name, unit=n-20, &
            status="replace", iostat=ios)
      if (ios /= 0) then
         print*, n, ios
      else
         write (unit=n-20, fmt="(i3)") n
         close (unit=n-20, status="keep")
      end if
   end do

end program open_files
```

Learn more about internal I/O.